Note that more than 80 dB of isolation is provided between the Tx ports at the center frequency, and as we measure response above and below the center frequency the isolation decreases. Thus, we determine that the hybrid must be tuned to half-way between the combined frequencies to secure reciprocal isolation between them.

The "PI" network provides the function of pulling the lines and the antenna system to resonance, referred to the hybrid load termination. Note that the power from each transmitter is split between the antenna and the hybrid load termination. This yields a 3 dB loss of power from each transmitter to the antenna plus connector and line losses of 0.2 to 0.4 dB depending on the operating band of the hybrid.

LOAD TERMINATIONS

A load termination must provide a constant resitive load over wide ranges of frequency and temperature, to effectively dissipate R.F. power as heat. The "perfect" load termination would show a constant resistance from D.C. to Daylight, under all conditions of thermal level. In practice, terminations designed for the two-way communications system applications are expected to have a VSWR of under 1.1:1 up to 1 GHz over their designated dissipation range. Wacom Products load terminations meet this requirement with margins of heat dissipation beyond their specified ratings.

CABLES AND CONNECTORS

In combiner systems, we deal with isolation exceeding 100 dB, and R. F. leakage can be quite detrimental to controlling these orders of decoupling. Only the best grades of cable and connectors can be expected to provide protection against R. F. leakage or inadvertent coupling in such systems. It is important that double shielded cables are used in every situation where flexible cable is required. All Wacom Products combiner systems employ double shielded cables such as RG-142/U, RG-214/U, RG-223/U, RG-402/U, etc.

Connectors are also important. It is still common in the two way industry to use UHF type connectors above 100 MHz. Our findings are that these connectors are marginal in characteristic at 150 MHz., poor at 450 MHz. and impossible at 850 MHz. We use Type N or BNC fittings for all transmitter or receiver multicoupling circuitry. These connector types have much superior VSWR above 100 MHz and in addition, have an inner contact sleeve in addition to the outer ferrule to reduce R.F. leakage from the connector.

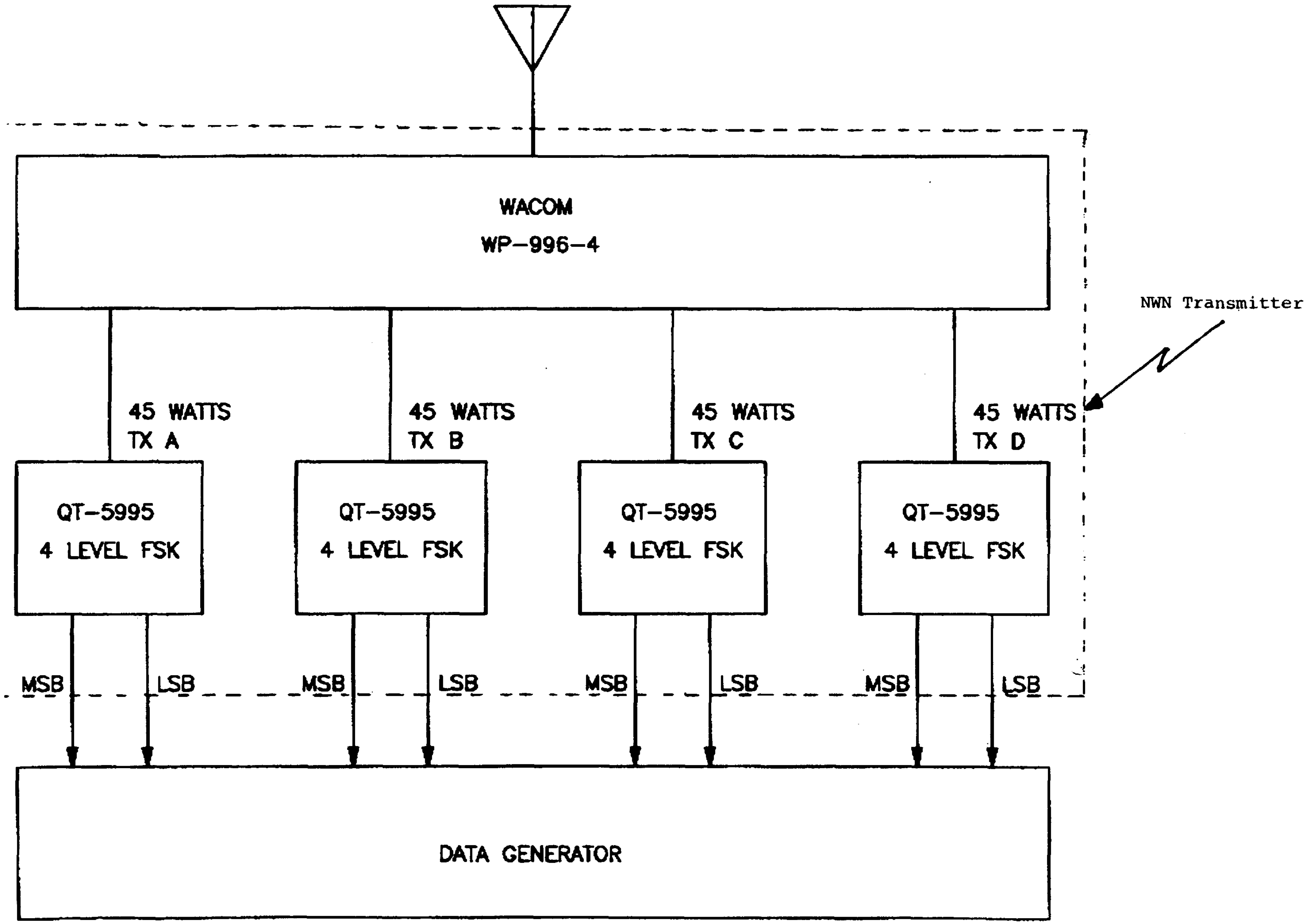

Fig. B-1

BLOCK DIAGRAM
TEST TRANSMITTER
11-25-92

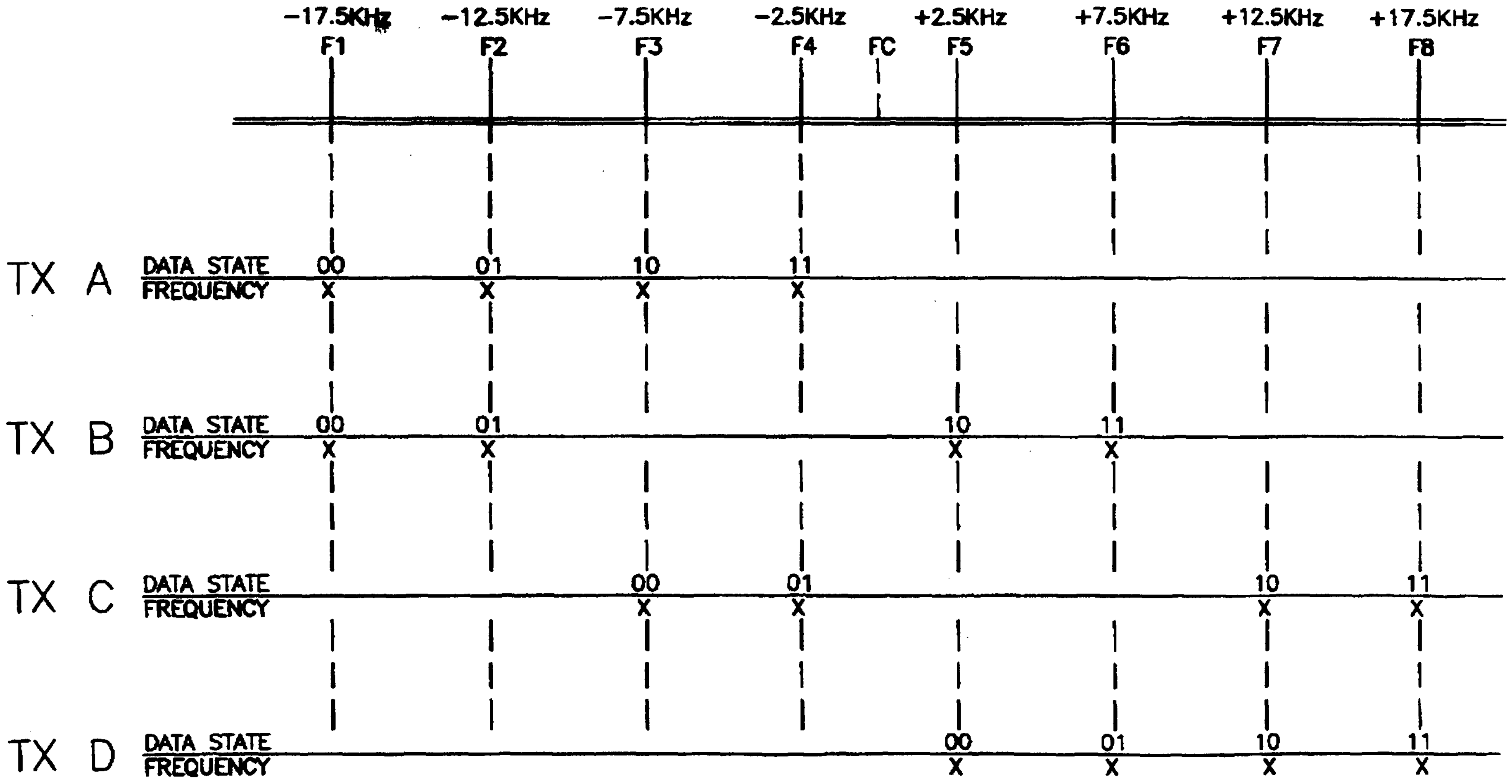

Fig. B-2

FREQUENCY SCHEME
TEST TRANSMITTER
11-25-92

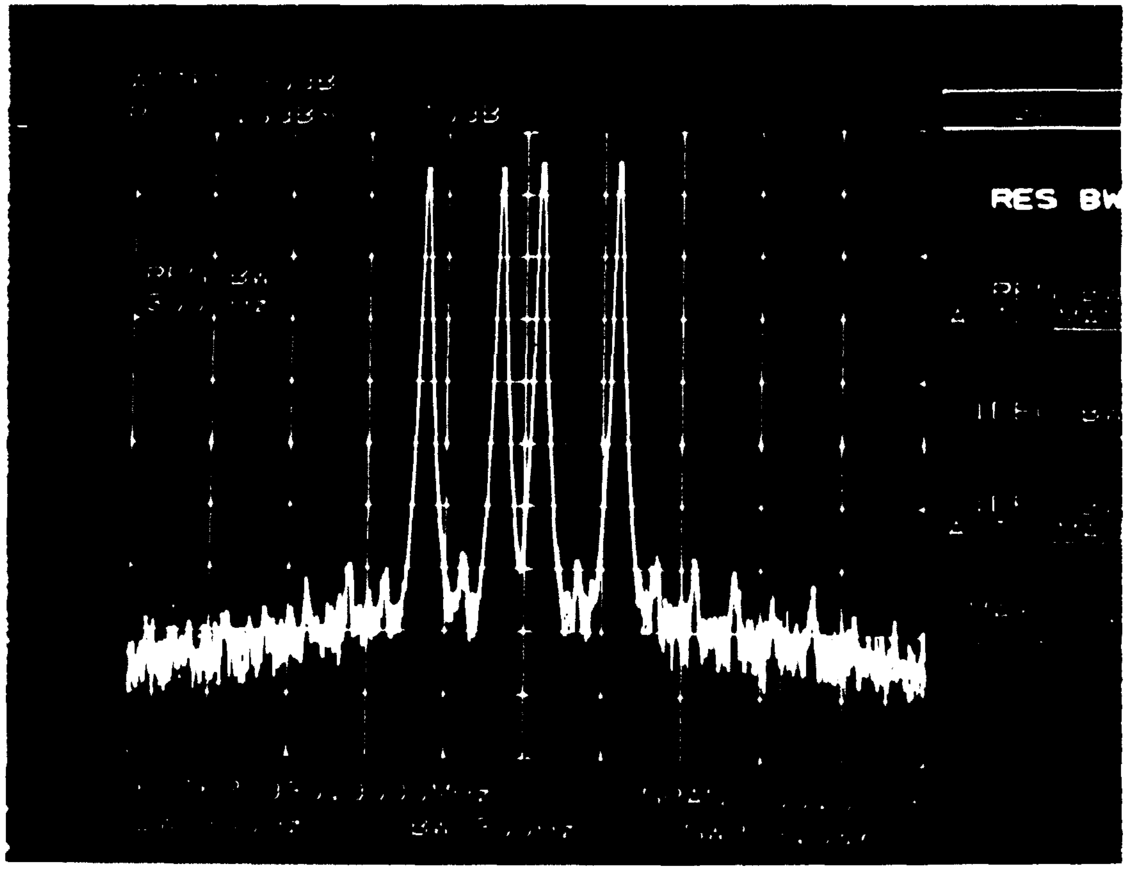

Figure B-3

COMPOSITE TRANSMITTER OUTPUT SPECTRUM
WITH FOUR UNMODULATED CARRIERS

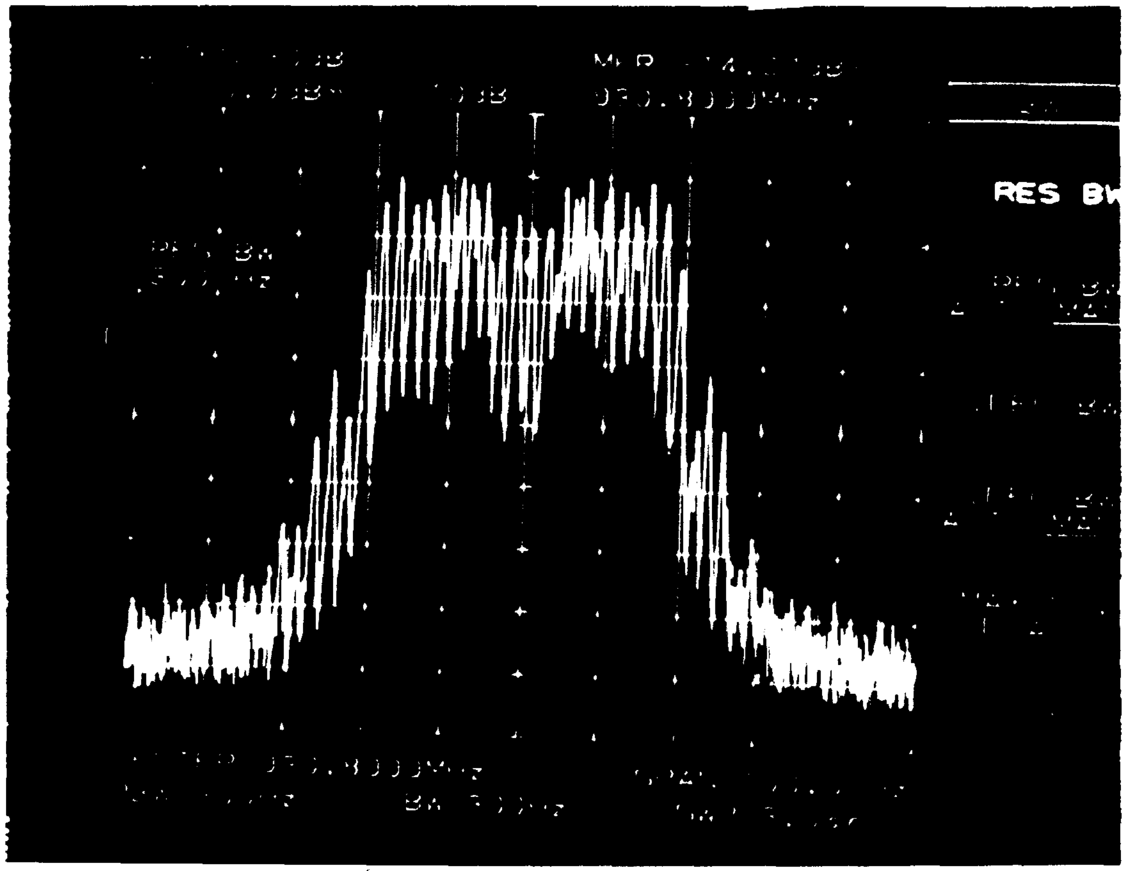

Figure B-4

COMPOSITE TRANSMITTER OUTPUT SPECTRUM WHEN ALTERNATING BETWEEN THE TWO PFSK SYNCHRONIZATION SYMBOLS

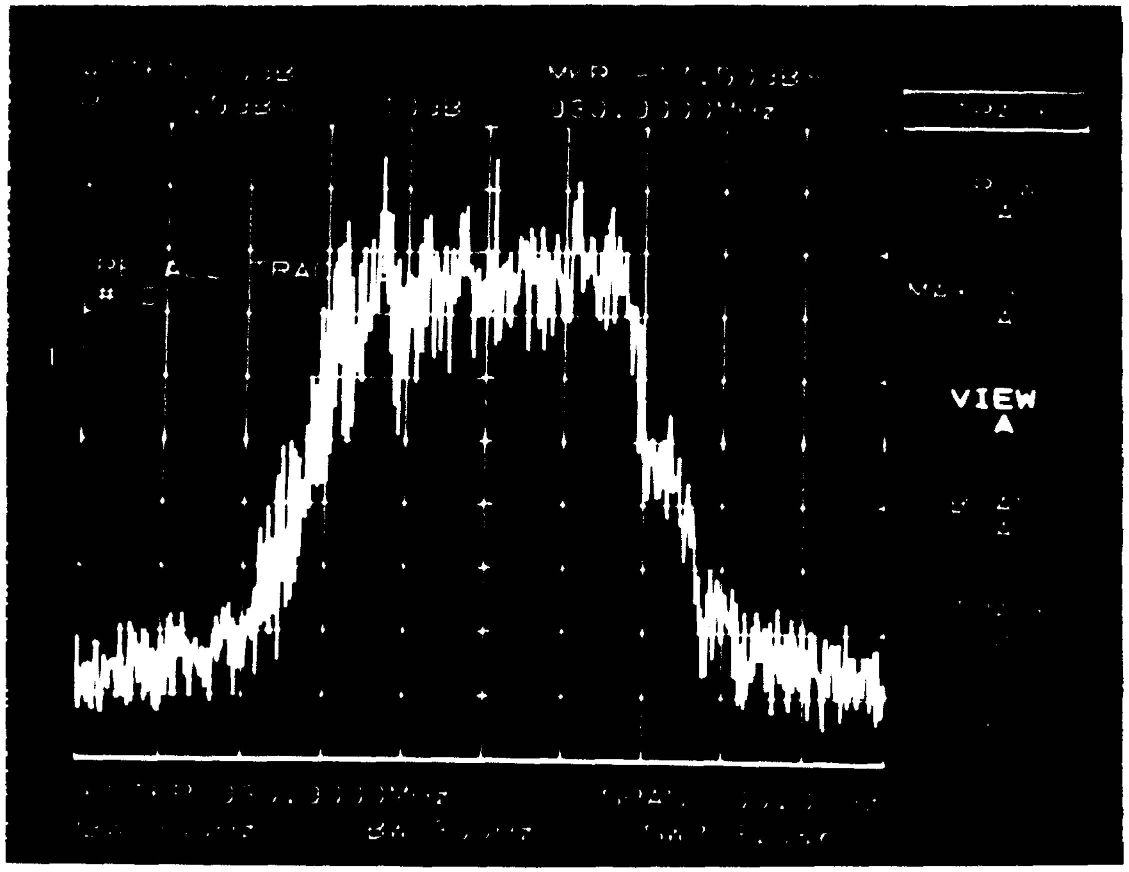

Figure B-5

COMPOSITE TRANSMITTER OUTPUT SPECTRUM WHEN SEQUENCING THROUGH ALL 64 PFSK SYMBOLS

TAB C

APPENDIX C

MTEL/WIRELESS ACCESS

NWN Receiver

Mtel Technologies, Inc.
200 South Lamar Street
Jackson, MS 39201

Wireless Access, Inc.
125 Nicholson Lane
San Jose, CA 95134

RECEIVER RF SECTION

An overall block diagram of the receiver section is shown in Fig. 1. Readily available off-the-shelf connectorized components were used to construct the receiver which was mounted on a plywood board for transportability. The size of the receiver was 12" by 18" not including the LO sources which were either HP 8656B or Marconi 2022D synthesized signal generators.

FRONT END AMP AND FIRST DOWNCONVERTER

The front end section was designed to yield good noise performance and adequate desensitization and image rejection. The first high pass filter protects the first broadband amplifier from saturation and spurious intermodulation products due to high power low frequency signals such as TV and FM radio. At least 40 dB attenuation is provided to these low frequency signals. The second bandpass low loss saw filter yields image rejection and further protects the mixer and IF amplifiers from spurious intermodulation and saturation due to frequencies outside the band of interest. The saw filter had a 3 dB bandwidth of 40 MHz and ultimate rejection of 25 dB. A total of approximately 30 dB of image rejection was provided by both the bandpass and high pass filters. A double-balanced passive mixer provided low side downconversion to a 45 MHz first IF frequency. An overall gain of 23 dB was measured from the antenna port to the input of the first mixer.

FIRST IF AND SECOND DOWNCONVERTER

The first IF frequency was chosen to be 45 MHz mainly due to the availability of off-the-shelf crystal filters with the appropriate bandwidth to pass all 8 frequency tones (35 KHz total bandwidth) without sacrificing system sensitivity (see below). In addition to setting the system sensitivity, the crystal filter eliminated spurious products and provided image rejection for the second downconversion. The four-pole crystal filter had a 3 dB bandwidth of 55 KHz and ultimate rejection of more than 50 dB. An attenuator was incorporated into the IF section to pad the amplifier gain to the level needed to maximize the dynamic range. A test point was incorporated into the second IF in order to monitor the strength of incoming signals during the field tests. This assured that the dynamic range of the receiver would not be exceeded. A second double-balanced mixer converted the 45 MHz signal to a final IF of 47.5 KHz.

SECOND IF

The second IF frequency was chosen to be high enough to allow adequate image rejection for the second downconversion and low enough to keep the sampling rate of the A/D within reasonable limits. The low pass anti-aliasing filter was designed to have minimal attenuation at the highest tone frequency (65 KHz) and drop to near maximum rejection at half the sampling rate (80 KHz). The rejection for frequencies above 80 KHz was 40-50 dB. In order to provide an adequate termination for the anti-aliasing filter, the output of the receiver was terminated in 50 Ohms. This termination also provided a predictable impedance level for calculation of the output voltage levels into the high input impedance of the A/D converter. A coupling capacitor blocked any DC offsets due to imbalances in the second mixer.

SENSITIVITY AND NOISE FIGURE

The noise bandwidth for the system is given by

$$B = g(f)^2 df \,/\, g_a^2$$

where g(f) is the transfer function of the system and ga is the maximum value of the transfer function. For this receiver, the noise bandwidth is determined by the crystal filter bandwidth. Substitution of the measured filter transfer function into Eq. 1 yields a noise bandwidth of 50 kHz. The equivalent input noise power of the receiver can be determined by setting the output signal to noise ratio equal to 1 and is given by

$$P_{nin} = kTBF$$

where F is the receiver noise factor. Using noise bandwidth calculated above in conjunction with the receiver noise figure of 3.6 dB, we obtain Pnin = -123.4 dBm at room temperature. The receiver sensitivity is given by

$$S = (P_{nin}(S/N)_{out}(4\pi^2 R_L \,/\, \lambda^2 G))^{1/2}$$

where R_L is the 50 ohm input impedance, $(S/N)_{out}$ is the required S/N ratio at the output required to achieve the intended bit error rate (assumed to be 9.5 dB), λ is the wavelength, and G is the receiver antenna gain (3 dBi). The sensitivity is then calculated to be 6.2 μV/M.

DYNAMIC RANGE AND INTERMODULATION PERFORMANCE

The total gain for the system was about 58 dB. Automatic gain control was not necessary due to the high dynamic range of the receiver design. Maximum voltage levels into the A/D are +-2.75 V which translates into a minimum detectable voltage of 84 μV or a total power level of -71.5 dBm into 50 Ohms. If the system gain is subtracted from this amount, the minimum detectable input power of -129.5 dBm is calculated which is below the noise floor of -123.4 dBm calculated above. The 6 dB margin was therefore incorporated into the design to insure appropriate amplification of all signals above the noise floor and to allow for quantization noise . Although the dynamic range of the A/D converter is 90 dB, the overall dynamic range of the receiver is limited to about 50 dB by the compression and intermodulation performance of the second mixer.

The intermodulation performance of the receiver was measured to be -14 dBc at the output with four equal strength tones at a power level of -72 dBm input to the receiver front end. This would allow for a selective fade of one of the input tones of more than 10 dB with the receiver input power near the top of the intended dynamic range. The intermodulation performance of course improves when the received signal strength is lower in the receiver's dynamic range. The intermodulation performance is of small consequence if all tones fade equally since only the four maximum strength tones are detected.

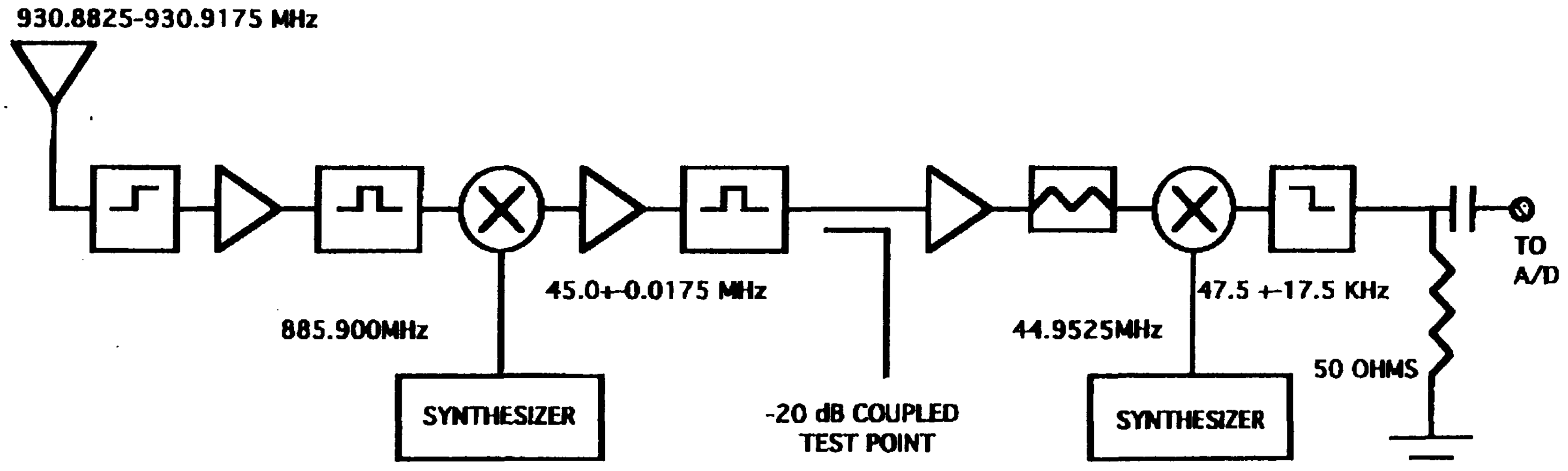

Fig. 1 Overall block diagram of receiver RF section.

RECEIVER - DIGITAL SECTION

This part of the document describes the digital section of the NWN Receiver. First a description of the hardware is provided and the design procedure involved in its choice. Following the hardware description, the software running on the hardware platforms is described.

HARDWARE DESCRIPTION

The digital section of the receiver consists of an Analog to Digital (A/D) converter box, followed by a PC plug-in board consisting of a digital signal processor (DSP) and accompanying hardware. A PC plays host to this hardware. A schematic diagram of the hardware and PC is shown in Figure 1.

A/D CONVERTER BOX

The first step required in the digital section of the receiver is the analog to digital conversion of the signal at the output of the RF front-end. The RF front-end design is such that the four of eight frequencies in the transmitted signal are present, after the second downconversion, at frequencies from 30kHz to 65kHz at intervals of 5kHz. In order to adhere to the Nyquist criterion, the sampling rate of the A/D conversion must be greater than twice the highest frequency (65kHz), which translates to 130kHz. However, in the digital processing (particularly the FFT processing) to follow, a frequency resolution of 5kHz is also necessary to discriminate among the frequencies. As a result, a sampling rate of 160kHz was chosen to simultaneously satisfy both criteria.

To maximize the dynamic range of the receiver, a sixteen bit A/D was chosen, which theoretically could provide a dynamic range of 94dB. An A/D box (BB71) from DSP Research, Inc. in Sunnyvale, California, was chosen which met the above mentioned requirements. The BB71 may also be programmed for any sampling rate, to up to 200kHz, through an external clock and DIP switches, providing us flexibility in future development. The BB71 can handle a peak voltage of +/- 2.75 volts into a 1kΩ input impedance.

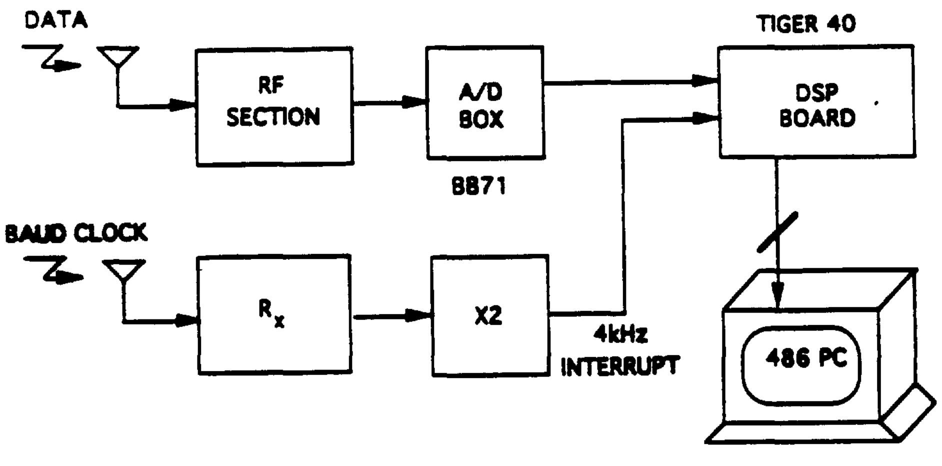

Figure 2: System Layout of NWN Receiver

DSP DEVELOPMENT BOARD

To perform all the digital signal processing following the A/D conversion, a PC plug-in board with a Texas Instruments TMS320C30 floating point DSP was chosen. The TMS320C30 is a powerful 32bit general purpose DSP capable of processing speeds of up to 40.0 millions of floating point operations per second (MFLOPS). A floating point processor also provides a large dynamic range and therefore precludes any possibility of problems encountered in a developmental system due to the limited dynamic range of a fixed-precision DSP. The TMS320C30 also has an optimizing C compiler allowing for most of the software development to be done in C, instead of the processor assembly language, thus cutting down development time. The plug-in board (Tiger 40) was also obtained from DSP Research to minimize any possible communication problems between the A/D box and the DSP board.

The Tiger 40 board has 512 kilobytes of zero wait state static RAM for program and data storage. The DSP board also has features to communicate with the host PC through dedicated registers, or dynamically mapable shared memory. This provides a high bandwidth channel between the DSP and PC for information transfer. The Tiger 40 board communicates with the A/D box (BB71) through a serial port on the DSP processor to which the BB71 is connected via a serial cable.

The Tiger 40 can also be input with external interrupts. This facility was used to input the baud clock received from the transmitter. The receiver for the baud clock and the 4kHz interrupt thus generated is also shown in Figure 2.

HOST PC

To provide the user interface to the NWN receiver implemented on the DSP board, a 50 MHz, 486 based PC with 8MB RAM, and a 200MB hard disk was chosen. This provides enough processing speed to communicate with the DSP board in real-time (at the baud rate of 4kHz), and at the same time provide a real-time display of test results. The amount of memory was chosen, keeping in mind, the high volume of data that the test measurements might generate. For example, consider storing data for a thirty second test. Given that ten floating point quantities are stored per symbol received, the storage requirements works out to approximately 1.2MB (with no compression). Hence, even if it is assumed that 100 MB of disk space is available just for data storage, data from about eighty-three test runs could be stored before it would be necessary to transfer them to floppy disks.

DSP SOFTWARE DESCRIPTION

The heart of the processing for the digital section of the NWN receiver, is performed on the DSP board (Tiger 40). In this section, the software running on the Tiger 40 board required to implement the NWN Receiver is described. A separate section is devoted to the software running on the PC which provides an interface to the DSP software.

First, a skeletal pseudo-code description of the DSP software is provided to clarify program flow, and the processing performed is detailed. Then the various software modules, which facilitate such a program flow, are described.

PSEUDO-CODE DESCRIPTION

The following is a pseudo-code description of the DSP software. Curly braces ({ }) are used to denote the beginning and end of loops. Each task performed is numbered and closed with a semicolon(;). There are also two interrupt routines which are not shown as part of this main processing loop as they run asynchronously with this main loop. These routines will be described in detail when the software modules are described. It will also be shown how these interrupt routines tie in with the processing performed in the following main loop.

```
Handshake with PC (initiated by PC);
begin (initiated by PC)
{
        initialize internal variables;
        intialize interrupt routines;
        begin test
        {
                i)      Check if baud-adjustment command from PC;
                if yes
                        make adjustment;

                ii) Check if test-to-end command from PC;
                if yes
                        exit;

                iii) Wait for baud timing interrupt (@4kHz);

                iv) Read appropriate thirty-two (32) data samples from main data buffer;

                v)  Perform FFT (magnitude calculation) of 32 data samples;

                vi) Store FFT bins of interest (8 out of 16) to be communicated to PC;

                vii) Sort FFT bins of interest and also their indices (1 through 8);

                viii) Use sorted indices to decode current transmitted symbol using symbol
                        table;

                ix) Pass current symbol to PC;

                x)  Pass energy in frequency bins, baud adjustment parameter & serial
                        number of current symbol to PC;
        }
}
end
```

SOFTWARE MODULES DESCRIPTION

To facilitate the above described program flow, the following software modules are implemented, each of which will be described in the following subsections :

i) Host Command Module

ii) Direct Memory Access (DMA) Interrupt Module

iii) The Baud Timing Interrupt Module

iv) The Buffer Management Module

v) The Fast Fourier Transform (FFT) Module

vi) The Symbol Detection Module

vii) The PC Communication Module

Host Command Module

This module interprets commands from the host PC and takes appropriate action (steps (i) and (ii) of the main loop). In particular, the baud adjustment commands are accumulated in the variable **baud_offset**. This value is not changed beyond the range of -41 to 0 for reasons which will be explained in a following section. Also, a special character from the PC indicates the end of a test, on the receipt of which, the DSP program exits.

DMA Interrupt Module

As mentioned earlier, the samples from the output of the A/D converter are available in the DSP's serial port. The Direct Memory Access module, implemented as an interrupt routine, reads the samples from the serial port of the DSP processor and stores them in the main data buffer (mentioned in task (iv) of the pseudo-code) for further processing. This data buffer is 1600 samples long. The DMA module fills the buffer, then interrupts the CPU letting it know that the buffer is full. This is very useful as costly processor cycles are not used to interface with the A/D box. Given an input sampling rate of 160kHz, a sample is available from the A/D every $6.25\mu s$. However, the processor is interrupted only once every 10ms ($1600*6.25\mu s$), where 1600 is the length of the buffer.

After the buffer is full, the DMA module circles back and begins filling samples again from the first location in the buffer. Consequently, this main data buffer is implemented as a circular buffer of length 1600 (about 40 symbols) on the DSP.

Baud Timing Interrupt Module

The **baud timing interrupt module** is the second interrupt routine implemented. This routine utilizes the 4kHz external baud clock available from the transmitter to mark baud edges in the main data buffer where samples are written into. When an interrupt is caused by the 4kHz clock, the current location where the DMA module is writing samples into is stored in a variable (**this_baud**). This provides a mark for where a nominal baud has ended. Such a mark is also available from a previous interrupt (**last_baud**), resulting in a window of data, nominally between **last_baud** and **this_baud**, which can be processed. The two variables, **this_baud** and **last_baud**, also provide an elegant mechanism for "waiting for an interrupt" required in step (iii) of the main module, as their being unequal implies the arrival of a new interrupt. This point will be clarified further in the next section.

Buffer Management Module

The buffer management Module collects the appropriate window of data (32 samples) from the data buffer beginning at the location (**last_baud+baud_offset**)(task (iv) in the main loop). It accounts for the data buffer being circular, and wraps around if needed, to read the correct samples. Note that the current accumulated value of the offset (**baud_offset**), as communicated by the PC in step (i) of the main loop, is used in picking this appropriate window here. Note that this **baud_offset** is only allowed to be negative. Such a limitation on the value of **baud_offset** guarantees that only samples which have already been stored by the DMA module in the data buffer are processed. This **baud_offset** value can be between -41 and 0, providing a potential phase adjustment of up to 360 degrees @ ~4kHz.

As forty (40) samples are processed per baud on an average, the buffer management module provides data at the nominal baud rate of 4kHz (160/40) to the DSP for further processing. The buffer management module along with the **baud_offset** variable provides a reliable mechanism for reading the thirty-two valid samples within the forty samples corresponding to each baud.

After the appropriate window of data is read, **last_baud** is set equal to **this_baud** to prevent the possibility of reading the same data again. Note that this also means that the value of **last_baud** and **this_baud** will remain equal to each other until another 4kHz interrupt occurs. This fact is utilized while waiting for an interrupt in step(iii) of the main loop.

The module also converts the 16bit integer values provided by the A/D to the floating point values required by the DSP. It further checks that enough buffer space is available for the DMA routine to write samples into. In this way, the DMA and baud timing interrupt routines along with the buffer management module provide a seamless interface between the A/D box and the DSP processor.

FFT Module

The FFT **module** implements the basic baud level processing necessary to detect the four of eight frequencies that are present in any particular baud (steps (v),(vi) and (vii) of the main loop). Given a sampling rate of 160kHz and a nominal baud rate of 4kHz, forty (40) samples per baud (in other words a time window of 250μs) are available for processing. However, given the switching time of the transmitter between baud, only a 200μs (or 32 samples) window is assumed to have all the four frequencies faithfully present. Consequently, an FFT size of 32 naturally fits in with the processing rates available. Also the advantage of FFT processing would be lost if the number of samples was not a power of 2. Note that this is the reason that the buffer management module reads thirty-two samples from the data buffer each time. It is assumed that the 32 samples read from the data buffer by the buffer management module are the valid data samples within the forty data samples corresponding to a symbol. Also note that the FFT size of thirty-two provides the exact frequency resolution of 5kHz (160/32) necessary in the receiver.

The FFT is performed in-place, utilizing the fact that the data input is real. Thus the frequency domain information is available in exactly the same locations where the time-domain real data was, when originally input. Normally, even if data is real, intermediate values in the butterfly calculations within a FFT calculation may be complex, thus potentially requiring more storage locations than the original real data (thirty-two). However, at each butterfly stage in the implemented algorithm, symmetries associated with the FFTs of real data are exploited to maintain storage requirements at thirty-two, the length of the real data. The bit reversal portion of the FFT calculation has to be performed first, thus implying a decimation in time algorithm, to exploit these symmetries. This algorithm results in the most efficient radix-2 FFT algorithm for real data, in terms of both storage and computation. The details of the algorithm may be found in the June, 1987 issue of the IEEE Transactions on Acoustics, Speech and Signal Processing which is attached as Appendix A.

Note that a 32 length FFT of real data will provide frequency information in sixteen (16) frequency bins. However, only eight frequency bins are of interest. As a result, the magnitude squared (energy) associated with only these eight frequency bins is calculated resulting in a further saving of computation. These eight frequency bin energies are then stored so that they can be passed on to the PC for storage on disk.

The FFT module, after calculating the energy in the eight frequency bins sorts them and their indices in ascending energy order to facilitate symbol detection.

Symbol Detection Module

Once the four highest energy frequency bin indices are determined, the symbol detection module decodes the particular symbol transmitted using these four highest indices as a key into a lookup table (step (viii) of the main loop). The symbol lookup table is used as a header file by the DSP software and is present in the file DEC-SYM.H, which is present in the \t30\nwn_code directory.

PC Communication Module

The PC Communication Module transfers information from the DSP for real-time display of test measurements and also storage for off-line processing and analysis. The parameters that are transferred are the current symbol received (through a data transfer register) in step (ix) of the main loop, and the energies in the eight frequency bins in the current baud (through shared memory) in step (x). Additionally, the value of **baud_offset** is sent to the PC for real-time screen display for the benefit of the user, and a serial number for each baud is provided so that the PC can check if it is collecting all the symbols sent over by the DSP. In terms of the digital processing required to implement the NWN Receiver, only the actual symbol error rate calculations (if in fact symbol error rate calculations can be considered an integral part of a receiver) are performed on the host PC.

It should be mentioned here that all the above mentioned modules are implemented in C, except for the actual FFT calculation, which is implemented in C30 Assembly, to meet real-time constraints. Using on-chip timers on the DSP the real-time requirement of the above processing was determined to be between

175-185ms. Note that about 250ms of real-time is available for each baud processing.

HOST (PC) SOFTWARE DESCRIPTION

In this section, the software running on the host PC which interfaces with the DSP software is described. As with the DSP software, first a pseudo-code description of the software is provided, followed by a description of the modules which enable such an implementation. Then a detailed description of the test modes supported is provided. This includes an enumeration of the possible tests that a user might run, a way of changing the configuration of a test for each of these modes, and what information the user might hope to obtain in each of these test modes.

PSEUDO CODE DESCRIPTION

The following is a pseudo-code description of the host software. The nomenclature used is similar to that used to describe the DSP software.

```
begin
{
        i)   Handshake with DSP;
        ii)  Start off DSP software;
        iii) Initialize default variables;
        iv)  Set up main menu screen;
        v)   Wait for user choice (from keyboard);

        if(choice = comments)
        {
                vi) Take comments for test;
                vii) Go back to (iv);
        }

        if(choice = type of test)
        {
                viii) Set up test type;
                ix) Go back to (iv);
        }

        if(choice = configuration of test)
        {
                x)  Change configuration parameters;
                xi) Go back to (iv);
        }

        if(choice = test (start test))
        {
                xii) check (type of test);
                case( type of test = waterfall)
```

```
                    {
                            xiii) Display received symbols from DSP;
                            xiv) Display value of baud_offset;
                            xv) Take keyboard input from user;
                            if("+" or "-")
                                    xvi) Pass to DSP;
                            if("ESC")
                                    xvii) Go back to (iv);
                    }

                    case(type of test = tune receiver)
                    {
                            xiv) Display value of baud_offset;
                            xviii) Perform symbol error rate (SER) calculations on
received                            symbols and display results;
                            xv) Take keyboard input from user;
                            if("+" or "-")
                                    xvi) Pass to DSP;
                            if("ESC")
                                    xvii) Go back to (iv);
                    }

                    case(type of test = SER calculation)
                    {
                            xviii) Perform symbol error rate (SER) calculations on
received                            symbols and display the results.
                            xv) Take keyboard input from user;
                            if("ESC")
                                    xvii) Go back to (iv);
                    }


                    case(type of test = timed data collection)
                    {
                            xix) Take symbols and energies provided by DSP and store in RAM;
                            xx) Display task being done;
                            xxi) After time out transfer symbols and energies from RAM to hard
disk;

                            xv) Take keyboard input from user;
                            if("ESC")
                                    xvii) Go back to (iv);
                    }
            }

            if (choice = quit)
            {
                    xxii) Disable DSP software;
            xxiii) Exit;
        }
}
end
```

HOST SOFTWARE MODULES

The host (PC) software broadly consists of the following modules which are described in the following subsections :

i) Application Control Module

ii) Screen Display Module

iii) SER Module

iv) Storage Module

v) Access Module

<u>Application Control Module</u>

This module is the main shell of the program, which controls the host program (NWN.EXE under DOS), and the DSP software. It sets up the handshake with the DSP and starts the DSP software running (tasks (i) and (ii)). It sets up the default configuration settings (task (iii)) for a test, reads the frame delimiter character from the NWN.CNF file, and the characters that are in the frame for possible symbol error calculations. It then uses screen routines which are part of the Screen Display module to set up the main menu screen (task (iv)). It then waits for input from the user (task (v)). It then interprets commands from the user and starts appropriate tasks as is shown in the pseudo code.

If the user decides to put in some comments for the test, these are recorded by the application module (task(viii)) and made available to the storage module when it stores data for this particular test. If the user decides to change the type of test, the application module records that change (task(viii)). Similarly, if the user decides to change the configuration of a test, the application module records that change (task(x)). When the user decides to quit, it disables the DSP software and exits gracefully (tasks(xxii) and (xxiii). Note that the DSP software is running all the time that the user is running the application (NWN.EXE under DOS). The advantage is that the parameters input by the user during a test remain valid for the entire duration of that test. This is particularly important for the **baud_offset variable** which the DSP uses in locating the correct window of data for FFT processing as explained earlier.

While in each of the test modes, if an ESC key is hit by the user, the application control module goes back to displaying the main menu, and waiting for further user input (task(xvii)).

In the waterfall and tune receiver modes, a "+" or "-" key input from the user is correspondingly conveyed to the DSP (task (xvi)) so that the value of **baud_offset** can be modified.

Screen Display Module

This module contains all the routines necessary to input data to the user screen. This includes setting up the main user screen (task(iv)), and all the other screens during the various test modes. In the waterfall mode, this module displays the received characters from the DSP continuously on the screen - eighty characters every second (task (xiii). It also displays the current value of **baud_offset** as echoed by the DSP (task(xiv)). In the tune receiver mode, it again displays the value of **baud_offset**, and SER calculations (task (xviii)). In the SER calculation mode, the module only displays the results of the SER calculations. In the timed data collection mode, it only displays the task being done - either writing to RAM or writing to hard disk.

SER Module

This module is used by the application control module to perform symbol error calculations, keep a track of frame slips and bad symbols in the tune receiver and SER calculation modes (task (xviii)). For symbol error rate calculations to be performed a frame delimiter character is necessary in the transmitted signal. This frame delimiter character is available in the NWN.CNF file which the application module reads at the beginning of a test. Similarly the entire expected frame is also read when the test is set up. This information is then available for the SER module to use.

The calculations are performed on a per frame basis after the expected delimiter character. If a delimiter character is not detected, the entire frame prior to the expected delimiter character is discarded, and no symbol error calculations are performed. In terms of performance, this is recorded as a frame slip. As a result, information about bad symbols within a slipped frame are not available to the user. When a frame delimiter character is detected, the just received frame prior to the delimiter character is compared with the expected frame, and the number of bad symbols is recorded. A running sum of bad symbols, frame slips and total number of symbols received is maintained by this module, and passed to the screen routines periodically for display. If n is the running sum of symbols that are in error among N total symbols received, the symbol error rate (SER) is calculated as

$$SER = n/N$$

Storage Module

This module is used by the application control in the timed data collection mode. The frequency bin energy information received from the DSP and the current symbol are stored on a 7MB RAM disk configured on the PC (task (xix)). This was necessary to meet the real-time constraint of storing frequency bin energy information at a 4kHz rate. The frequency bin energy values are stored as floating point quantities.

Once all the data for a particular timed test have been stored on the RAM disk, this data is then transferred to hard disk in a certain format (task (xxi)) using compression to save on disk storage, for off-line processing and analysis. On the disk, specific information about the test, as set up by the user in the test configuration, the possible comments etc. are also stored. This puts a particular stamp specific to a test on all collected data. This, it is hoped, would make archiving test data quite convenient.

Access Module

A separate program can be used to access the stored frequency bin information on hard disk. At the present time, the only mode supported is the on-screen display of the symbol, followed by the eight frequency bin energies present in that symbol. This program(RESULTS.EXE under DOS) is not callable from the application control module (or main menu), and has to be called separately from DOS. This routine does the decompression of the stored data on the fly. Typing "DIR DAT" from \NWN directory provides a complete listing of files stored in such a format.

All of the PC software is implemented in Borland C.

TEST MODES

As enumerated in the psuedo-code of the PC software there are four possible test modes. Each of them is described in greater detail in the following subsections.

Waterfall Mode

In the waterfall mode, a real-time display of received characters is available to the user. The host program merely displays the symbols received from the DSP on the user screen as characters. Eighty characters to a line are displayed after which the display of a new line begins. This mode continues indefinitely until interrupted by the user. Because of the overhead required to display symbols on the user screen, only one line of symbols (80) is displayed per second. The remainder of the symbols (4000-80=3200) are not displayed. In this mode, the **baud_offset** variable is also displayed on the upper left-hand corner of the screen. The user can change the value of this variable using the "+" and "-" keys on the keyboard. The PC communicates these key strokes to the DSP board which actually changes the value of the variable and echoes this change back to the PC. Now this changed value of **baud_offset** is used by the DSP in its processing. Using this aid, the user is able to move the window of processing around until a 'decent' scroll of symbols appears on the screen.

Receiver Tuning Mode

The value of baud_offset may also be similarly changed using the "+" and "-" keys in the receiver tuning mode. However, in this mode a real-time display of the symbol error rate, frame slips and bad symbols is displayed on the screen for the user's benefit. The value of **baud_offset** if of course also displayed. Here, the user is able to move the window of processing around until the symbol error rate, frame slips and bad symbols reach a desired value. This is essential before any timed tests performed are considered meaningful. Note that once the value of **baud_offset** is adjusted, and the user does not quit the user interface, this **baud_offset** value remains the same for the remainder of tests as explained earlier. Thus the user can first tune the receiver using either of the two mentioned modes (waterfall or receiver tuning), and then perform any actual symbol error rate tests and/or timed data collection. Eventually this tuning mechanism will be automatic and transparent to the user and probably performed entirely within the DSP.

Symbol Error Rate Mode

This mode is a subset of the Receiver Tuning mode where only the symbol error rate, frame slips and bad symbols are displayed for the user's benefit. The user does not have the capability of changing the **baud_offset** value. This mode is best used after the receiver has been tuned and the **baud_offset** value has been set to the correct value. Then the user is in a position to let the receiver run for any desired length of time and determine what the performance is.

Timed-data Collection Mode

In timed data collection mode, the user has the ability to store symbols, and the energies present in each of those symbols for off-line analysis. Three time levels are supported : 5, 10 and 30 seconds. Again this mode is best used only after the receiver has been tuned. In this mode, the user is only made aware of what the PC host program is doing (collecting data or storing data). Symbols received and the frequency bin energy information of that symbol are stored in RAM first (in real-time) and then stored to the hard disk (in non real-time). This is necessary to meet real-time constraints of the 4kHz symbol rate. No other information about the test is available to the user in this mode. This mode is well suited to performing off-line analysis on the energy present in the various frequency bins in the received symbols. Another useful feature in this mode is that the file associated with each test is user configurable. A set of comments about a certain test may also be incorporated in the stored file, thus providing the user some reference points while accessing that data later for analysis. Compression is used while storing these files to maximize use of disk space. The filenames reflect the configuration parameters associated with a certain test making the files easily identifiable. Each filename label is of the type "Txx-yy-z.NWN", where :

xx	=	location of test
yy	=	test I.D.
z	=	repetition number of test

To access these files later, a separate utility RESULTS.EXE under DOS is available as described earlier. This utility can provide an ascii output of the information in the files, which the user than might use to analyze the received data further.

TAB D